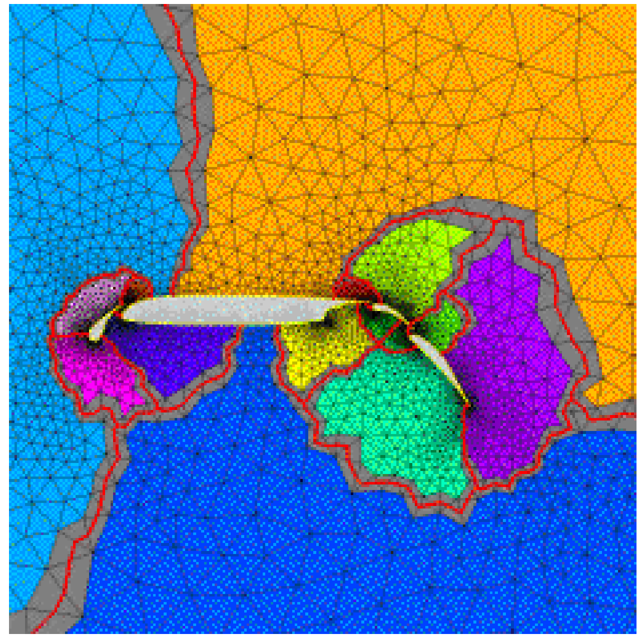
January-February 1996, Vol. 2, No. 15



NEWS

Metacenter Offers `Tremendous Potential' for SP2 Users

By Elisabeth Wechsler

What began as simple discussions between NAS and Langley Research Center (LaRC) about IBM SP2 utilization has grown into a broader model for sharing emerging parallel systems resources at the two sites. Called a "metacenter," this model uses computer resources at multiple locations as "one logical entity," explained Mary Hultquist, lead SP2 systems analyst at NAS.

"The metacenter approach has tremendous potential for users -- more computing power and more flexibility in running their jobs," commented Judith Utley, senior parallel systems administrator at LaRC, in Hampton, VA. For example, users can submit parallel jobs to a single scheduler, which sends each job to the most appropriate SP2 at either site.

To minimize communications overhead, smaller jobs generally would be directed to the 48-node SP2 at LaRC, while larger jobs would be sent to the 160-node SP2 at NAS. There are exceptions, however, because although the systems are compatible in terms of system software and applications, each will offer different usage options -- for example, specific memory requirements or the need for a HiPPI driver on a node.

First Step -- Similar Configurations

The metacenter concept evolved from discussions last spring about how to get more users on the SP2 at LaRC. The NAS SP2, with a large Cooperative Research Agreement (CRA) user group, was experiencing slow job turnaround, while the LaRC SP2 had no CRA user accounts and much lower utilization, explained Geoff Tennille, Manager of the HPCC testbed at LaRC.

"We wanted to increase our SP2 utilization. The first step was to ensure that the SP2 systems at LaRC and NAS were similarly configured," Tennille said. The major difference was that LaRC was using IBM's LoadLeveler product for job scheduling, while NAS was utilizing its own Portable Batch System (PBS). At the time, LoadLeveler permitted interactive access to the SP2, while PBS did not. Once PBS was modified to allow interactive jobs, it was installed on the SP2 at LaRC. Using PBS to schedule jobs provides a rudimentary metacenter for those users validated on both systems, he explained. In addition, PBS provides file staging, so that users don't have to duplicate their data files on each system.

To achieve the first step of system compatibility, NAS and LaRC coordinated efforts for an operating system upgrade (AIX 4.1.3) for both SP2s, which was completed in the fall. NAS Facility staff helped with the upgrade at LaRC, with LaRC staff assisting in the NAS upgrade a few weeks later, combining their expertise to improve the process.

Both systems are funded by the NASA High Performance Computing and Communications (HPCC) Program through CRAs. The testbed project, which began in 1994 (see "IBM SP2 Delivers Performance, Challenges to Consortium Partners," November-December '94), involves a group of government, university, and commercial sites. Under the agreement, each site has its own research objectives and was not required to coordinate systems at this level of detail. Eventually, however, tight budgets and limited resources paved the way for the broader metacenter objective.

"Doing the upgrades together gave us a rare opportunity for training, plus it was cost effective," Utley said. Users of each SP2 were able to run their jobs at the other's site during the upgrades.

Next -- Solve Policy, Configuration Issues

The second step for the metacenter involves resolving administrative and configuration problems, such as improving the distributed file system, increasing disk capacity, combining user accounting methods, and coordinating user support, Tennille explained.

Weekly teleconferences are held between the NAS and LaRC staffs to facilitate solutions, and the two groups met numerous times at Supercomputing '95 in San Diego. Some of the short-term issues to be worked out include policies for queue limits, scratch file space, migrating software, and training.

Individual staffing at both NAS and LaRC is still required because of the commitment to pathfinding efforts at both sites. "These are two different machines that can't be managed remotely -- staff with day-to-day familiarity is needed because each system is independent and has unique aspects," Hultquist said. "Each SP2 is being tinkered with as a testbed, and each manager needs to know what's been done on a day-to-day basis. It doesn't make sense for either site to take charge of the other's machine."

Tennille observed: "While it's no big deal to use a Cray [supercomputer] remotely, with the SP2 it's more problematic. You need some control of the system configuration to experiment with different approaches."

Technological Challenges Still Ahead

The two supercomputer sites will continue working closely together, so that users can run jobs on either system as seamlessly as possible.

In order to achieve that goal, some technological issues will have to be overcome, including reducing the user access bottleneck by implementing Distributed Computing Environment/Distributed File System

(DCE/DFS), which is already in use on the LaRC testbed cluster of IBM RS/6000 workstations.

"DFS will be faster, smarter, and more reliable. DCE brings several services including secure remote procedure calls," explained Mark Smith, IBM technical liaison working at NAS.

In addition, message-passing software will need to be changed. "PVM (Parallel Virtual Machine) is not as efficient on the SP2 because it doesn't take advantage of the IBM switch technology," Smith said. MPI (Message Passing Interface) is more suited to this diverse metacenter concept because it's the emerging standard acknowledged by the industry. In the area of software applications to explore, High Performance Fortran (HPF) is near the top of the metacenter's list, he added. NAS now has three HPF compilers on the SP2, from Portland Group, APR, and IBM.

"Both systems are pushing their respective technological limits," Hultquist said. "The metacenter is good for users -- because of quicker turnaround and more resources -- and good for the staff -- who draw from a larger base of expertise."

- More information

Next Article | Contents | Main Menu | NAS Home

NAS Benchmarks Evolve to `Keep Pace' With Industry

By Bill Saphir

A new source-code implementation of the NAS Parallel Benchmarks, known as NPB 2.0, was released at the December 3-8 Supercomputing '95 Conference in San Diego. NPB 2.0 is the first step in an ambitious plan to ensure that the NAS benchmarks keep pace with changes in the parallel computing industry. The new version supplements -- rather than replaces -- NPB 1.0.

While NPB 1.0, the original version of the benchmarks, continues to provide an important measure of parallel system performance, NAS recognizes that the benchmarks need to evolve to remain effective. The NPB must address the following changes in parallel computing:

- General-purpose parallel computer architectures are converging; for instance, the last three parallel testbeds at NAS have been distributed-memory MIMD (multiple instruction, multiple data) machines based on RISC processors.
- Increasing standardization makes source code compatibility possible; there are now two distributed memory parallel programming standards: Message Passing Interface (MPI) and High Performance Fortran (HPF).
- Parallel systems users in general, and NAS users in particular, are writing portable codes that are intended to be run on several architectures with little tuning.
- Parallel computers and applications are getting bigger.
- New classes of applications are becoming an important part of the NAS workload. These include multidisciplinary, multizonal, and multiblock applications, and unstructured and adaptive solvers.

What NPB 2.0 Is -- and Isn't

NPB 2.0 is primarily an MPI-based source code implementation of five benchmarks in the NPB 1.0 suite (the five benchmarks are: BT, block-tridiagonal; FT, FAST Fourier Transforms; LU, lower-upper diagonal; MG, multigrid kernel; and SP, a brief explanation of the original NPB suite).

These implementations, which are intended to run with little or no tuning, measure the performance that an average user might hope to obtain for a portable code. In contrast, NPB 1.0 implementations show what can be obtained through careful machine-specific tuning.

Performs `Poorly' on Vector Supercomputers

NPB 2.0 is targeted toward distributed memory parallel computers based on RISC processors with cache-based memory hierarchies. NBP 2.0 codes are optimized neither for long vector lengths nor for Cray-style memory systems. As a result, NPB 2.0 performs rather poorly on traditional vector supercomputers and should not be used to compare them directly with RISC-based machines. However, NPB 2.0 shows that codes designed for RISC systems may not perform well on vector architectures -- and vice versa.

Figures 1 and 2 show preliminary NPB 2.0 results obtained on the SP2 and SGI PowerChallenge array at NAS, a CRAY T3D at the Jet Propulsion Laboratory, and an Intel Paragon at Langley Research Center. Figure 1 compares the performance of the NPB 1.0 (vendor-optimized) and NPB 2.0 implementations on several architectures and illustrates the potential benefit of careful, architecture-specific tuning.

Figure 2 shows the per-processor Mflop/s rate for the five NPB 2.0 implementations. Because NPB 2.0 codes run unmodified, it is meaningful to count floating point operations and calculate Mflop/s (million floating point operations per second) rates. This is not true for NPB 1.0 codes, where different implementations may have drastically different operation counts. The chart confirms what parallel system users already know -- that actual floating point performance is usually well below peak performance -- and that percentage of peak performance varies widely among machines.

NPB 3.0 Components

NPB 2.0 was the first step in the plan to expand the scope of the NAS Parallel Benchmarks. Three remaining components to the plan, which will comprise NPB 3.0, are:

- release MPI/F77 implementations of the remaining three benchmarks. These are: EP (embarrassingly parallel), CG (conjugate gradient) and IS (integer sort)
- release HPF implementations of the entire set
- develop new benchmarks to reflect new applications being run at NAS

Finishing the implementations of the final three benchmarks, is a relatively low priority. These benchmarks were not included as part of NPB 2.0 primarily due to resource limitations, though there are some technical reasons for not including EP, and possibly CG.

Releasing HPF implementations of the benchmarks is very important to evaluating whether HPF is a viable programming paradigm on current parallel architectures. Until recently, there was a lack of robust HPF compilers with which HPF benchmarks could be developed and tested -- moreover, the immaturity of the compilers would have guaranteed poor performance on HPF benchmarks.

HPF compilers are just now becoming mature enough to make NPB implementations feasible. While NAS has begun internal development of HPF benchmarks, resource limitations preclude a full-scale development effort. To solve this problem, NAS has solicited HPF implementations from the vendor community, and will use those implementations as the starting point for unbiased standard implementations of the benchmarks. Tentative plans call for work to begin early this year. Whether or not HPF can be competitive with message passing, and in what areas, remains to be demonstrated.

New Benchmarks Most Time-consuming

The third element in the plan, developing new benchmarks, is the most time- consuming of the NPB plans. The original benchmarks were designed to reflect the NAS workload in 1991. Since then, several new types of applications have taken an increasing share of CPU space on the NAS parallel systems. These include multizonal and multiblock codes, multidisciplinary applications, and unstructured and adaptive computations. And, since NAS was recently named the lead center for high- performance computing within NASA, it is likely that NAS computers will soon run applications from outside the computational aerosciences domain. The NPB must be able to adequately reflect these disciplines, as well.

Along with NPB 2.0, these NPB 3.0 elements should keep the NAS benchmarks an important part of parallel computer benchmarking into the next century.

Production Codes Needed

NAS is currently deciding what types of applications should be reflected in the new benchmarks, and is looking for production codes that can be used as a starting point for scaled down pseudo-applications. If you have a code that is not represented in the NPB suite and which might make a good candidate for a NAS benchmark, send email to the NAS benchmark group at **npb@nas.nasa.gov**.

- Details on NPB 2.0, including a description of new class "C" sizes for benchmarks, as well as new reporting requirements for NPB 1.0 submissions.
- NAS Parallel Benchmarks Explained

Next Article | Contents | Main Menu | NAS Home

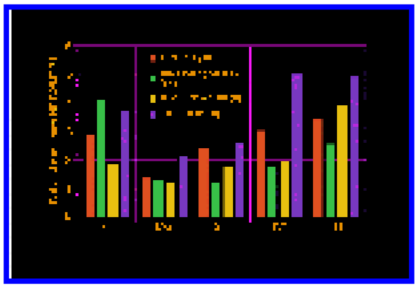
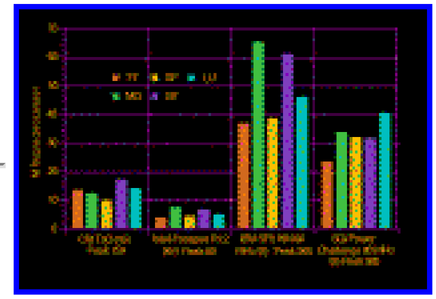
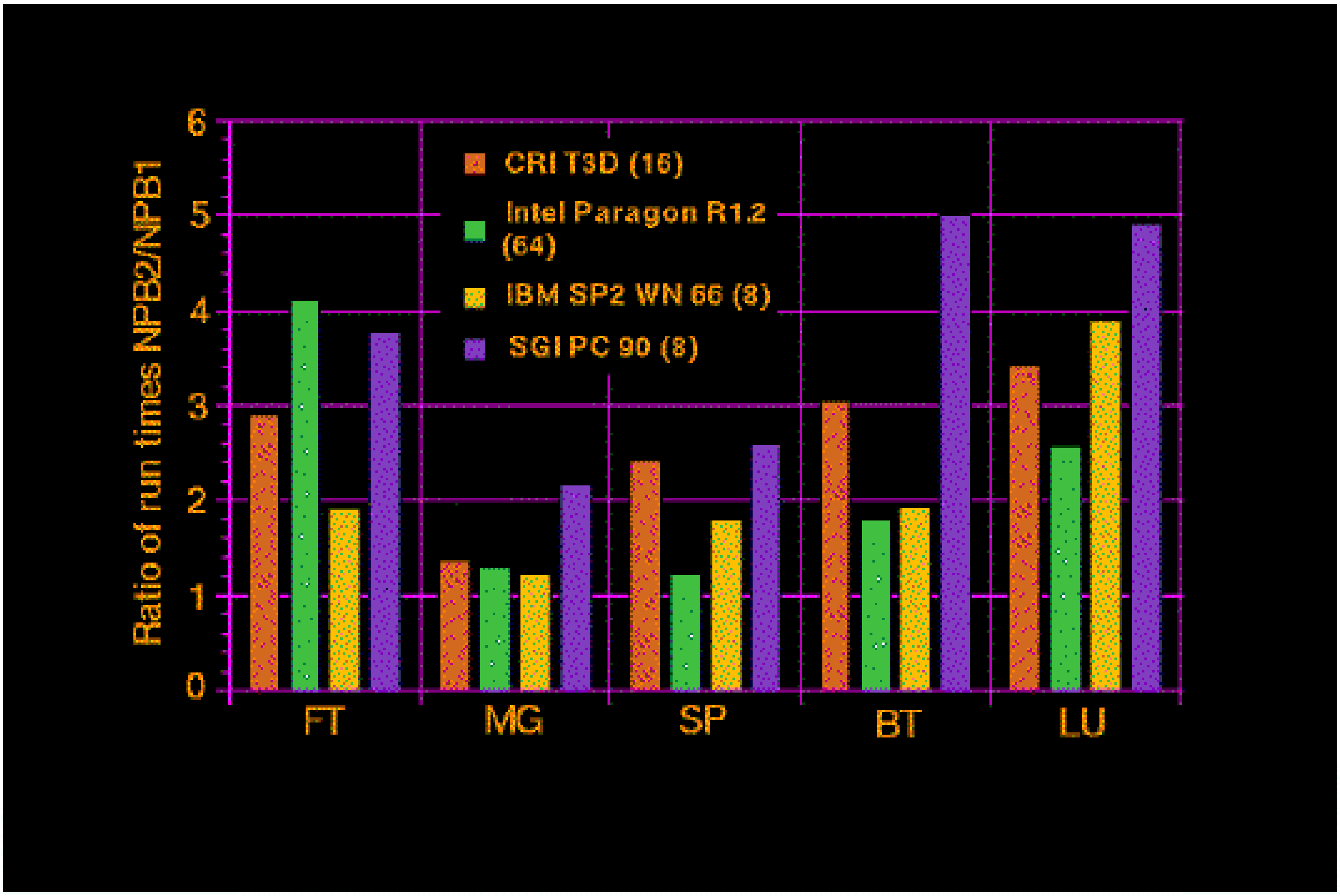


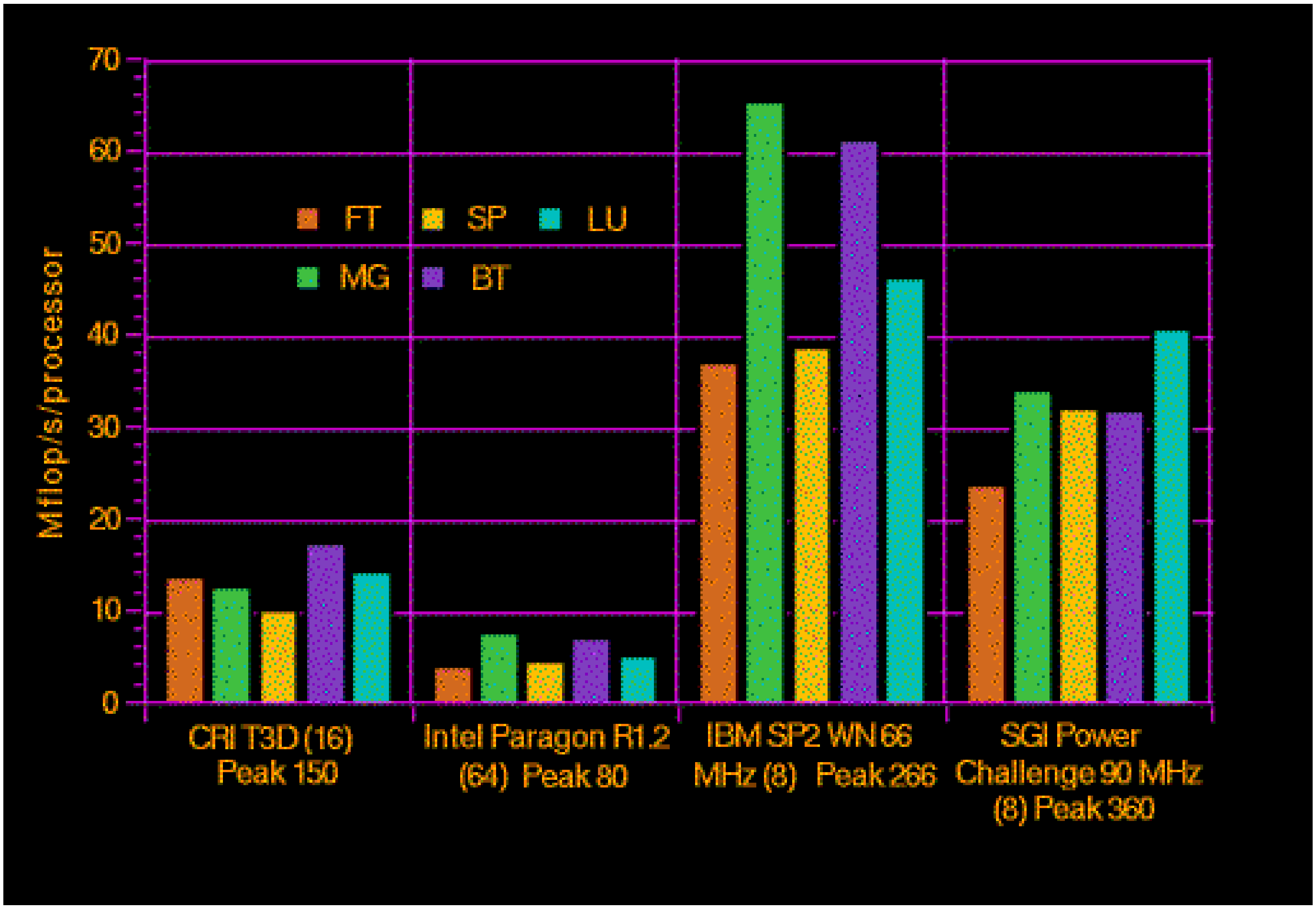
Figure 1 -- Ratio of NPB 2.0 time to NPB 12.0 time -- Class A.

Figure 2 -- Sample Mflop/s per processor on NPB 2.0 suite -- Class A.



[to the article.]


Ratio of run times NPB2/NPB1
6
5
4
3
2
1
0
CRI T3D (16)
Intel Paragon R1.2 (64)
IBM SP2 WN 66 (8)
SGI PC 90 (8)
FT
MG
SP
BT
LU


Mflop/s/processor
70
60
50
40
30
20
10
0
FT
SP
LU
MG
BT
CRI T3D (16) Peak 150
Intel Paragon R1.2 (64) Peak 80
IBM SP2 WN 66 MHz (8) Peak 266
SGI Power Challenge 90 MHz (8) Peak 360

Next Article | Contents | Main Menu | NAS Home

NAS Parallel Benchmarks Explained

To help clear up confusion about NPB 2.0, here's a brief explanation of the original NAS Parallel Benchmarks.

In order to understand NAS Parallel Benchmarks (NPB) 2.0 -- which have confused many in the high-performance computing industry -- a general explanation of NPB 1.0 is helpful. The original NAS Parallel Benchmarks, developed in 1991, are a suite of eight programs to measure the performance of parallel computers and to compare their performance with that of conventional supercomputers. Since then, the NAS benchmarks have gained wide acceptance in the high-performance computing community as a standard measure of supercomputer performance.

The NAS benchmarks are unique in their "pencil and paper" approach -- they are implemented by computer vendors based on detailed specifications in the NPB report. Vendors are free to choose algorithms and programming models appropriate for their different machines. The pencil and paper approach acknowledged fundamental barriers to portability in programming paradigms, languages and libraries, and in computer architectures.

For an explanation of the NPB 1.0 implementations, see "NAS Parallel Benchmarks Set Industry Standard for MPP Performance," NAS News January-February 1995.

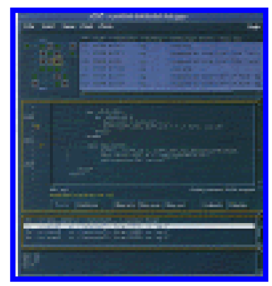
Portable Debugger Will `Make Life Easier' for Parallel Users

by Elisabeth Wechsler

p2d2, a NAS-developed portable distributed debugger for multiprocess programs, will soon make life easier for users of the IBM SP2 and Silicon Graphics Inc. cluster testbeds, predicts Robert Hood, senior research scientist in the NAS parallel tools group. The program was demonstrated at Supercomputing '95, held December 3-8 in San Diego, and is expected to be ready for beta users by February.

p2d2 employs a unique abstraction technique called "process navigation," which scales to a large number of processes and provides an overview of an entire computation, with the ability to zoom in on a single process, Hood explained.

"Historically, multiprocess debuggers have not presented the `big picture' very well," Hood said. "p2d2 helps isolate `misbehaving' processes. Without this sort of abstraction -- and if you have a lot of processes -- you'd be looking at a lot of windows."



When running p2d2, the debugging information is abstracted on three levels that can be viewed simultaneously:

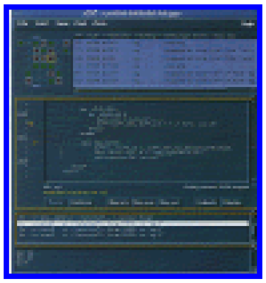
- A small amount (4 bits -- less than one character) of information about each process, represented by one of up to 16 colored symbols in a cell in the two-dimensional grid at top left of screen.
- One line of information about a handful of processes, in gray shaded area at top right of screen.
- Detailed information about a single process, lower half of screen.

"You can use p2d2 to establish which processes are waiting to receive a message, for example, and then look at the communications details for individual processes," Hood explained. "You could also use it to determine which processes have a data value outside an expected range. This approach would be suitable for debugging up to 256 processes," he continued.

Hood's future plans include investigating ways to tailor p2d2 for debugging CFD-specific programs -- a much-needed tool.

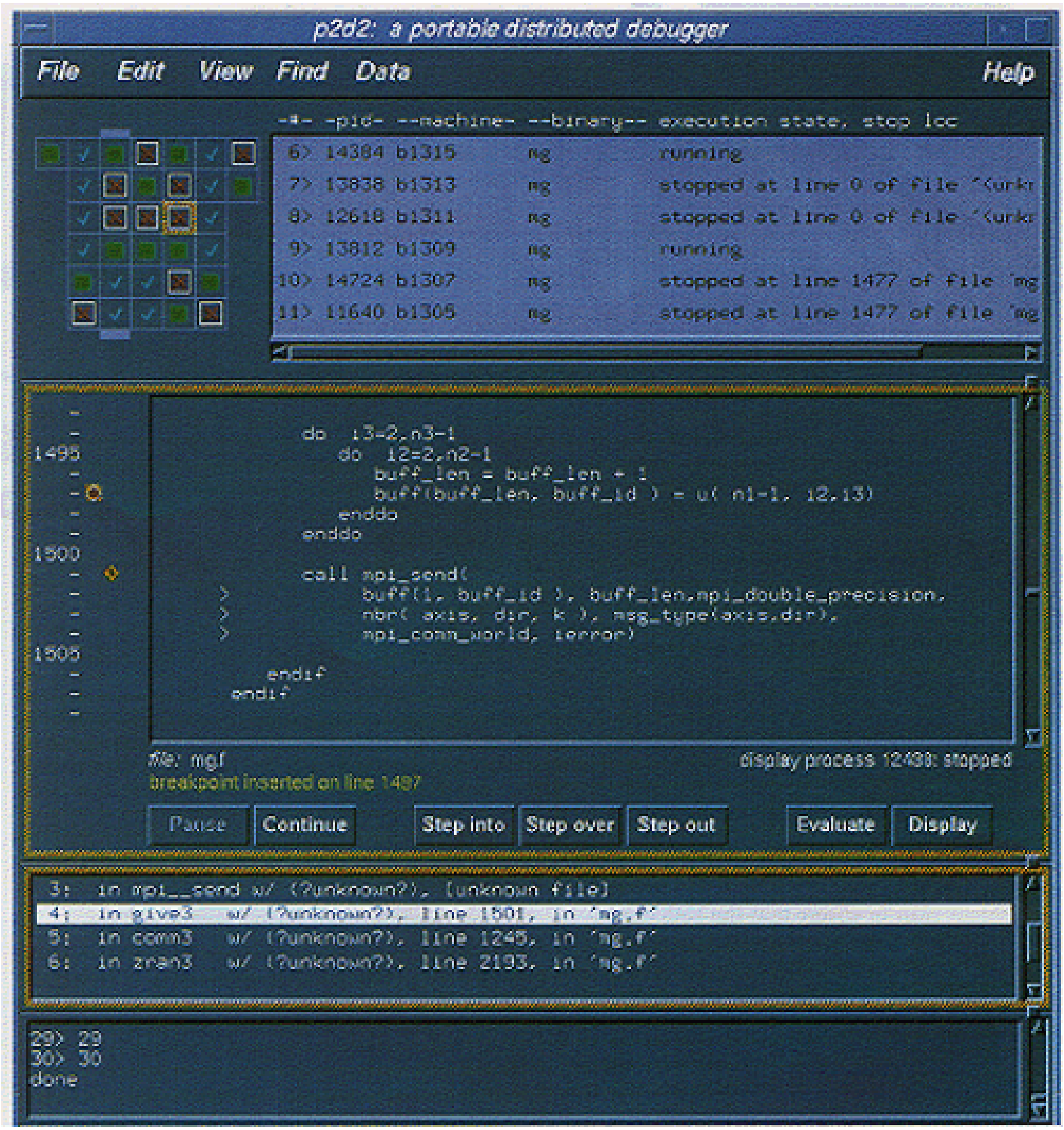
- More information, including the p2d2 architecture and related technical papers.

[Next Article] [Table of Contents] [NAS News Home Page] [NAS Home Page]



User interface for p2d2, the NAS portable distributed debugger for the IBM SP2 and SGI cluster testbeds. Three levels of debugging information are abstracted at top left, top right, and lower half of the screen. In this example, 33 processes are being debugged.

to the article.


p2d2: a portable distributed debugger
File
Edit
View
Find
Data
Help
-#- -pid- --machine- --binary-- execution state, stop loc
6> 14384 b1315 mg running
7> 13838 b1313 mg stopped at line 0 of file
8> 12618 b1311 mg stopped at line 0 of file
9> 13812 b1309 mg running
10> 14724 b1307 mg stopped at line 1477 of file
11> 11640 b1305 mg stopped at line 1477 of file
1495
1500
1505
do i3=2,n3-1
do i2=2,n2-1
buff_len = buff_len + 1
buff(buff_len, buff_id) = u(n1-1, i2,i3)
enddo
enddo
call mpi_send(
buff(1, buff_id), buff_len,mpi_double_precision,
nbr(axis, dir, k), msg_type(axis,dir),
mpi_comm_world, ierror)
endif
endif
file: mg.f
display process 12438: stopped
breakpoint inserted on line 1497
Pause
Continue
Step into
Step over
Step out
Evaluate
Display
3: in mpi__send w/ (?unknown?), [unknown file]
4: in give3 w/ (?unknown?), line 1501, in 'mg.f'
5: in comm3 w/ (?unknown?), line 1245, in 'mg.f'
6: in zran3 w/ (?unknown?), line 2193, in 'mg.f'
29> 29
30> 30
done

User-assisted Automatic Parallelization of Block-tridiagonal Solvers Reduces Wall-clock Time

by Bob Bergeron

NAS users who employ approximate factorization for implicit solutions to the Euler and Navier-Stokes equations frequently must solve systems of block-tridiagonal equations. Parallelizing the code implementing these solutions can significantly reduce program wall-clock time. However, even "automatic" parallelization for shared-memory systems can require significant user intervention. Techniques for effective automatic parallelization on block-tridiagonal solvers should also have general application to other NAS codes. This article describes how to automatically parallelize a shared-memory block-tridiagonal solver.

The Standard Algorithm

For realistic problem sizes, the application of central spatial differencing to the implicit three-point time differencing of the Euler conservation laws produces a large banded system of algebraic equations. Approximate factorization of the multidimensional Euler operator reduces the solution of this large system to sequential solutions of one-dimensional operators. The standard algorithm will solve this reduced system of equations, which displays a simpler block-tridiagonal structure.

To solve the block-tridiagonal system, shared-memory versions of the standard algorithm employ a Lower-Upper (LU) decomposition within a recursive block-Thomas algorithm. The solvers typically use the number of lines and the number of points per line to define the calculational plane.

For a plane containing L lines with P points per line, let a, b, and c denote, respectively, the subdiagonal, diagonal, and super block-tridiagonal matrices of the L-by-P block-tridiagonal system. The submatrices comprising the block-tridiagonal system will be M-by-M, where M is either 4 or 5 depending on whether the problem is two- or three- dimensional. Let f denote the right-hand side matrix and V denote the correct solution matrix. Both f and V are L-element M-component column matrices.

Each of the L lines containing P points has an a, b, c, and f matrix. The L-by-P system of M-by-M block-tridiagonal equations is represented as follows:

```
b1   c1   V1   =   f1
a2   b2   c2   V2   =   f2
          .    .    .
          ai   bi   ci   Vi   =   fi
               .    .    .
               aP-1  bP-1 cP-1 VP-1  =  fP-1
                    bP   cP   VP    =  fP
```

For simplicity, the indexing on the lines has been suppressed. Expressed schematically, the standard algorithm has the form:

(LU decompose the first block)

```
for point p = 1
LU decompose bp (find bp inverse)
f'p = bp-1 * fp for all lines for m = 1 to M
c'p = bp-1 * cp for all lines for m = 1 to M
end for loop on points
```

(Upper triangularize the matrix)

```
for point p = 2 to p = P
     f'p = fp - ap * f'p-1 for all lines for m = 1 to M
     b'p = bp - ap * c'p-1 for all lines for m = 1 to M
```

LU decompose bp for all lines for m=1 to M

```
     f'p = b'p-1 * c'p for all lines for m = 1 to M
     c'p = b'p-1 * c'p for all lines for m = 1 to M
end for loop on points
```

(Backsolve)

```
for point p = P
     Vp = f'p for all lines for m = 1 to M
end for loop on points
for point p = P-1 to p = 1
     V'p = f'p - c'p * V'p+1      for all lines for m = 1 to M
end for loop on points
```

Initial Restructuring for Parallelism

For each point on a line, this algorithm must have the LU decomposition completed on the previous point in order to visit the next point. The above scheme makes this demand explicit by pulling the first point out of the main loop. This requirement illustrates a recurrence (a loop construction in which the results from one loop iteration feed back into subsequent iterations of the same loop). Loops that display recurrences cannot be parallelized because their answers depend on a specific order of execution and the random processor completion prevents a specific order of execution; therefore, the algorithm on the index visiting the points cannot be parallelized.

However, an examination of the algorithm indicates that all loops have a common inner loop over all the lines and that this common inner loop has no recurrence. Inverting the loops to produce a recurrence-free outer iteration on the lines would allow parallelization. But to achieve speedups on a parallel vector computer, the restructured loop must maintain vectorization. Wall-clock time will not decrease when a vectorized loop is replaced by a restructured parallel scalar loop. Although this initial restructuring produces highly parallel code, the recurrence on the points will inhibit inner loop vectorization, and this restructuring will not perform faster than the standard algorithm.

Restructuring for Parallelism

The independence of the iteration over the lines also allows the inner loop iterations to be bundled into groups in such a way that processors can work on groups of lines in parallel. An outer loop, which is wrapped around the calculation to parcel work to each of the processors, has the Fortran form:

```
     lgroup = (L+ncpus-1)/ncpus
c ...execute this loop in parallel
      do 1000 ll= 1,L,lgroup
      lbeg = ll
     lend = min(L,ll+lgroup-1)
c ...visit groups of lines with this loop
      do 10 i = lbeg, lend
      ...
  10 continue
      ...
1000 continue
```

The maximum number of CPUs available to the job (ncpus) determines the number of parallel groups, and the number of lines (L) determines how much parallel work each group receives. Each of the inner loops that previously visited all lines in the plane will now visit groups of lines defined by lbeg and lend. Although this bundling decreases the vector length, it still manages to assign vector work to multiple processors.

Using Autotasking Directives

The Cray autotasker, FPP, requires a directive because the preprocessor cannot verify that the outer iterations (over the groups) are independent. The FPP directive NOSYNC advises the autotasker to ignore the results of its dependency analysis and parallelize the loop. This directive should be given to the autotasker only when you are absolutely sure of the parallelism in the subsequent loop. You can test the effect of the various directives on a Fortran source, file.f, by applying the preprocessor with the following:

```
fpp -l fppl file.f
```

Then, examine the listing file, fppl, for advice from the autotasker.

With the proper autotasking directive, the CRAY C90s (vonneumann and eagle) parallel algorithm has the final form: **CFPP$ NOSYNC** for all groups.

```
for point p = 1 for all lines in group
LU decompose bp (find bp inverse)
f'p = bp-1 * fp for m = 1 to M
c'p = bp-1 * cp for m = 1 to M
end for loop on points
for point p = 2 to p = P for all lines in group
f'p = fp - ap * f'p-1 for m = 1 to M
b'p = bp - ap * c'p-1 for m = 1 to M
LU decompose bp (find bp inverse)
f'p = b'p-1 * f'p for m = 1 to M
c'p = b'p-1 * c'p for m = 1 to M
end for loop on points
for point p = P for all lines in group
Vp = f'p for m = 1 to M
end for loop on points
for point p = P-1 to p = 1 for all lines in group
Vp = f'p - c'p* V'p+1 for m = 1 to M
end for loop on points
end for loop on groups
```

The improved restructuring allows the autotasker to produce a file for the midprocessor containing a microtasking directive of the form:

```
CMIC$ DO ALL SHARED(...)
```

around the outer loop to enable parallel iterations of this loop. Submitting this file to the compiler will generate the required object file for loading into the executable. The program will request multiple CPUs when the block-tridiagonal solver is entered.

Lessons for Autotasking

Creating efficient parallel programs on shared memory Cray architectures requires careful restructuring of algorithms to augment vectorization with parallelism. Frequently, the most obvious parallel structure will not reduce the program elapsed time because it distributes scalar work to vector processors. Effective parallel performance requires that the parallel regions consist of vector work, which is partitioned to multiple CPUs. The example described above indicates that an outer wrapper loop is an effective method for combining vectorization with parallelism. The FPP listing file is an extremely useful tool for identifying the type of work given to multiple processors and for expanding the ability of the autotasker to create parallel code. Occasionally, potential data dependencies may require that you override the autotasker analysis. The FPP listing file will usually suggest the appropriate directives.

The speedup of the parallel solver depends strongly on the amount of parallel work available to the multiple processors, and larger values of P and L lead to larger amounts of parallel work and greater speedups. A 4x4 block-tridiagonal test code requesting four CPUs in the **vonneumann** multitasking queue (mtasklg) displayed the speedups (ratio of the one-processor elapsed time to the four-processor elapsed time) listed in the table below.

Number of Points	Number of Lines	Speedup
48	48	0.350
192	192	0.721
336	336	1.020
432	432	1.208
576	576	1.448
720	720	1.710

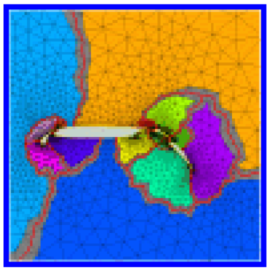
The measurements indicate that the numbers of points and lines must exceed 300 before the elapsed time of the parallel version falls below that of the one-processor version.

Source level versions of these parallel solvers are available to NAS users. A similar autotasking approach may be applied to the scalar pentadiagonal systems derived for time-dependent problems.

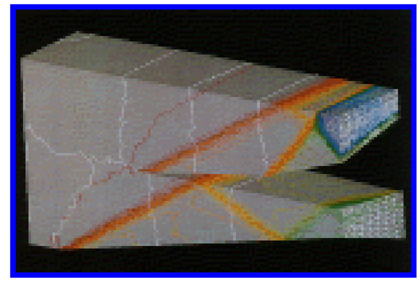
For more information about parallel solvers, send email to **bergeron@nas.nasa.gov**.

Next Article | Contents | Main Menu | NAS Home

Next Article | Contents | Main Menu | NAS Home



Ames CFD Research in Parallel Computing Highlighted at AGARD Lecture Series



By Marcia Redmond

Top computational fluid dynamics (CFD) researchers from the U.S. and Europe gathered to discuss advances and future trends in the application of parallel computing at the Advisory Group for Aerospace Research and Development (AGARD) Lecture Series on "Parallel Computing in CFD," held in October at NASA Ames Research Center. The conference included a presentation on parallel CFD algorithms by Ames researcher Tim Barth.

Walt Brooks, NAS Systems Division Manager, opened the AGARD series and called this year's participants a "pathfinding group, willing to take a chance" in today's computing environment. Brooks said, "NAS R&D and pathfinding [efforts] must focus on applications that are important in aerospace. The AGARD Lecture Series is ideal for us because it enables our key users to interact directly with other CFD researchers to integrate these latest pathfinding advances into the work we do here at NAS."

Brooks said that, since Ames has been named the Information Technology Research and Development Center for NASA, over the next ten years "the NAS Systems Division will take on a broader role in developing technology to support climate modeling, [simulation of] colliding galaxies, and spacecraft design in parallel to aircraft design," and added, "we are ready to make the transition." (See "NAS Will Play Pivotal Role in Agency Supercomputing Consolidation, NCITE," November-December 1995.)

Current research efforts in CFD have made it possible for scientists to take full advantage of the various available architectures by using specific algorithms for the simulation of fluid flow.

Tim Barth, research scientist in the Aeronautics Technologies Division at Ames, discussed basic parallel computing algorithms and procedures used in CFD, including implicit Newton methods for solving the compressible Euler and Navier-Stokes equations, and the pursuit of parallel scalability of these methods using domain decomposition techniques.

Newton Algorithms Invaluable

Barth emphasized the use of exact or approximate Newton methods in CFD. He said that "there are difficulties with existing CFD methods (such as computing aircraft fluid flow near wing stall conditions) -- and algorithms based on Newton's method can easily compute most of these flows."

In particular, Barth discussed a CFD algorithm that is under development, which utilizes the latest technology in unstructured mesh algorithms for the Euler and Navier-Stokes equations and is based on Newton's method. The algorithm includes a one-equation turbulence model for high-Reynolds- number aerodynamic computations. The flow equations are discretized and solved using Newton's method. To demonstrate the predictive capability of the method, he showed CFD examples of fluid flow about multiple-component airfoil geometries and compared them with experimental data. Barth also focused on techniques for extending this CFD algorithm to parallel computing architectures.

Schwarz Algorithm Subdomain Problems

As an introduction to parallel computing methods, Barth reviewed the Schwarz algorithm for elliptic equations -- a technique that requires the isolated solution of subdomain problems. He discussed the way in which the overlap of subdomains theoretically affects performance and parallel scalability. Barth showed that the Schwarz algorithm can be applied to the solution of hyperbolic equations by introducing the concept of "characteristic projectors," which prevent the overspecification of subdomain boundary data.

Barth also presented numerical examples for multiple-component airfoil geometries (see figure on front page), which show how inviscid and viscous fluid flow computations benefit -- in terms of improved rates of convergence- -from an increase in subdomain overlap, using the Schwarz algorithm.

According to Barth, as an alternative to the conventional Schwarz domain decomposition procedure, a Newton-Krylov technique can be used. In this approach, overlapped, incomplete lower-upper (ILU) factorizations are used to precondition a Krylov projection algorithm for solving the linear system of equations produced in Newton's method. Barth has demonstrated the efficiency of this method through inviscid and viscous computations. However, he cautioned that "although the domain-decomposed ILU preconditioning accelerated the algorithm, true parallel scalability was not achieved." In contrast, with another technique -- called domain decomposition substructuring -- Barth has achieved some promising numerical results for advection-diffusion model problems, in which parallel scalability was achieved.

New Solver for Computing Flow Fields

The IBM SP2 and Silicon Graphics Inc. (SGI) Power Challenge cluster housed at the NAS Facility provide the basis for much of Barth's parallel algorithm development and testing. Using the MPI (Message Passing Interface) protocol, Barth and colleagues at Ames are developing a three- dimensional Euler and Navier-Stokes solver, based on Newton's method. This parallel flow solver has successfully computed a variety of flows ranging from Euler flow in a generic supersonic jet intake (as shown in the figure below) to Navier-Stokes flow over a multiple-component wing. Both IBM and SGI chose Barth's parallel flow solver for interactive demonstrations at Supercomputing '95, held last month in San Diego.

While much progress has been made in increasing parallel performance curves for the 3D solver, Barth said that "substantial parallel algorithmic research is still needed to achieve efficient, scalable, and implicit algorithms."

- More information on Barth's work, including his paper, " An Unstructured Mesh Newton Solver for Compressible Fluid Flow and its Parallel Implementation" (3.7-Mb compressed PostScript).

Next Article | Contents | Main Menu | NAS Home

Jan - Feb 1996, Vol. 2, No. 15

NAS NEWS

Volume 2, Number 15 January – February 1996

Metacenter Offers 'Tremendous Potential' for SP2 Users

By Elisabeth Wechsler

What began as simple discussions between NAS and Langley Research Center (LaRC) about IBM SP2 utilization has grown into a broader model for sharing emerging parallel systems resources at the two sites. Called a "metacenter," this model sees computer resources at multiple locations as "one logical entity," explained Mary Hultquist, lead SP2 systems analyst at NAS.

"The metacenter approach has tremendous potential for users—more computing power and more flexibility in running their jobs," commented Judith Utley, senior parallel systems administrator at LaRC, in Hampton, VA. For example, users can submit parallel jobs to a single scheduler, which sends each job to the most appropriate SP2 at either site.

To minimize communications overhead, smaller jobs generally would be directed to the 48-node SP2 at LaRC, while larger jobs would be sent to the 160-node SP2 at NAS. There are exceptions, however, because although the systems are compatible in terms of system software and applications, each will offer different usage options—for example, specific memory requirements or the need for a SSPH drive on a node.

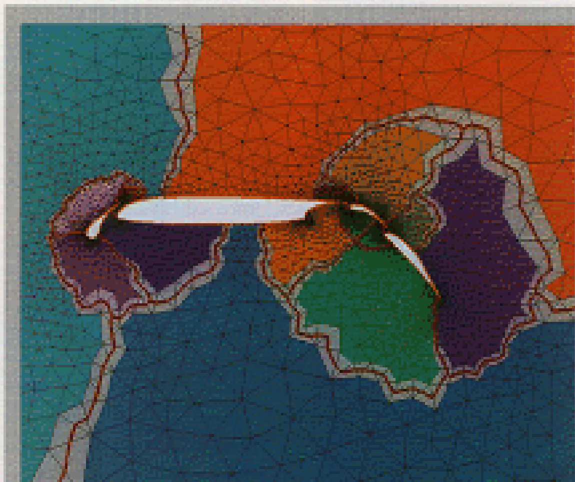
First Step—Similar Configurations

The metacenter concept evolved from discussions last spring about how to get more users on the SP2 at LaRC. The NAS SP2, with a large Cooperative Research Agreement (CRA) user group, was experiencing slow job turnaround, while the LaRC SP2 had no CRA user accounts and much lower utilization, explained Geoff Tennille, Manager of the HPCC testbed at LaRC.

"We wanted to increase our SP2 utilization. The first step was to ensure that the SP2 systems at LaRC and NAS were similarly configured," Tennille said. The major difference was that LaRC was using IBM's LoadLeveler product for job scheduling, while NAS was utilizing its own Portable Batch System (PBS). At the time, LoadLeveler permitted interactive access to the SP2, while PBS did not. Once PBS was modified to allow interactive jobs, it was installed on the SP2 at LaRC. Using PBS to schedule jobs provides a rudimentary metacenter for those users validated on both systems, he explained. In addition, PBS provides file staging, so that users don't have to duplicate their data files on each system.

To achieve the first step of system compatibility, NAS and LaRC coordinated efforts for an operating system upgrade (AIX 4.1.3) for both SP2s, which was completed in the fall. NAS facility staff helped with the upgrade at LaRC, with LaRC staff assisting in the NAS upgrade a

Continued on page 2



Example of a typical domain decomposition of a multiple-component airfoil geometry used in the parallel computation of fluid flow. Colored regions denote differing subdomains, while gray areas show mesh overlap. The image was part of a presentation by Tim Barth, entitled "Parallel CFD Algorithms on Unstructured Meshes," at the Advisory Group for Aerospace Research and Development (AGARD) Lecture Series, held in October. See article on page 3.

NAS Benchmarks Evolve to 'Keep Pace' With Industry

By Bill Saphir

A new source-code implementation of the NAS Parallel Benchmarks, known as NPB 2.0, was released at the December 3–8 Supercomputing '95 Conference in San Diego. NPB 2.0 is the first step in an ambitious plan to ensure that the NAS benchmarks keep pace with changes in the parallel computing industry. The new version supplements—rather than replaces—NPB 1.0.

While NPB 1.0, the original version of the benchmarks, continues to provide an important measure of parallel system performance, NAS recognizes that the benchmarks need to evolve to remain effective. The NPB must address the following changes in parallel computing:

- General-purpose parallel computer architectures are converging; for instance, the last three parallel testbeds at NAS have been distributed-memory MIMD (multiple instruction, multiple data) machines based on RISC processors.
- Increasing standardization makes source code compatibility possible; there are now two distributed memory parallel programming standards: Message Passing Interface (MPI) and High Performance Fortran (HPF).
- Parallel systems users in general, and NAS users in particular, are writing portable codes that are intended to be run on several architectures with little tuning.
- Parallel computers and applications are getting bigger.
- New classes of applications are becoming an important part of the NAS workload. These include multidisciplinary, multizonal, and multiblock applications, and unstructured and adaptive solvers.

What NPB 2.0 Is—and Isn't

NPB 2.0 is primarily an MPI-based source code implementation of five benchmarks in the NPB 1.0 suite (the five benchmarks are: BT, block-tridiagonal; FT, FAST Fourier Transform; LU, lower-upper diagonal; MG, multigrid kernel; and SP,

Continued on page 7

THIS ISSUE